Remember Me

Say it once. Ask for it later.

A voice and text memory assistant for iOS and Android

Rhythm Paudel

Project documentation — version 1.0

26 August 2026

Abstract

This report documents the making of Remember Me, a mobile application for iOS and Android that stores the small pieces of information a person would otherwise forget. The user says or types something they want to keep, asks for it back in ordinary words later, and is reminded about it when a reminder is what they actually wanted. The application was built to solve a problem most people already work around badly, using notes apps, screenshots and messages sent to themselves, none of which can answer a question.

This documentation covers the whole of the development, from the problem the application is meant to solve through to the finished system as it runs today. The report is made up of five main segments: introduction, background, development, the application in use, and conclusion. The application was developed under Agile methodology, and the sections of this report follow the order the work was actually done in.

A large part of the work described here is about the language understanding, because that is the part that decides whether the application feels intelligent or annoying. The report explains what the application does with a sentence once it has been understood, how it decides what matters and how hard to push, and why the finding and matching is done by the application itself rather than left to the user.

Additionally, several issues that could be faced after the deployment of the application, whether legal, social or ethical, have been discussed.

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

Remember Me is a mobile application for iOS and Android that keeps the small pieces of information a person does not want to lose. The user says or types something in ordinary words, and the application stores it. Later the user asks a question in the same ordinary words, and the application answers it. If the thing had a time attached to it, the application reminds them about it, and keeps reminding them if that is what the sentence meant.

Everybody already has a system for this, and almost nobody's system works. A phone number gets typed into the notes app, a prescription gets photographed, an appointment gets sent to a person's own chat window, and a car parking level gets repeated out loud in the hope of remembering it. All of these store the information. None of them can give it back when asked, because none of them understand what was written. The information is saved, and then it is lost inside the place it was saved.

The application was built around that single difference. Storing is the easy half. The half that matters is being able to ask for something back the way a person would ask another person, and to be told the answer, not shown a list of documents to search through manually.

1.1. Current Scenario

The amount of information a person is expected to hold in their head has grown much faster than any human memory has. A single adult is now expected to remember passwords for dozens of services, several account numbers, medication names and doses, appointment times, the birthdays of family and friends, expiry dates on documents, and where they have put things. None of this is difficult information. There is just too much of it.

The tools most people use for this were designed for a different job. A notes application is designed for writing, so it gives back a page of text. A photo library is designed for pictures, so a photographed licence becomes one more image among

thousands. A messaging app used as a notepad gives back a conversation. Every one of these can store, and every one of them makes the user do the finding themselves.

Voice assistants moved part of the way towards solving this. A person can already say a reminder out loud and have it set. What they cannot do is say an arbitrary fact and have it come back later in an answer, because those assistants treat a sentence as a command to be executed immediately rather than as information to be kept. Once the command is done, the sentence is gone.

1.2. Problem Statement

The main problem is that saving something and being able to find it again are treated as the same problem, when they are not. Every existing tool is good at the first and leaves the second to the user. The result is that people store information they never retrieve, and then behave as though they had never stored it at all, they look the number up again, or ask the person again, or pay for a replacement document.

The second problem is that a reminder and a fact are not the same thing, and neither are a reminder and a deadline. A person who says "the rent has to be paid by today" has not asked for a notification at one particular moment. They have stated a deadline, and what they want is to be told about it until it is done. A person who says "Rhythm’s birthday is on the twelfth of June" has not asked to be reminded every year at all, but would probably like to be. Most applications collapse these into a single alarm at a single time, which is why reminders are so often either useless or irritating.

The third problem is that the most valuable things a person would want to store are the ones they are least willing to hand over. A passport number, a card PIN, a prescription and a wifi password are exactly the information worth keeping and exactly the information nobody wants sitting in an ordinary note that syncs to a laptop and shows up in a search result. Any application that asks for this information has to be able to say clearly what happens to it.

The fourth problem is one the project ran into during development rather than one identified in advance, but it turned out to be the largest. Language is unbounded.

The number of ways a person can say "remind me until I have done it" is not a list that can be written down and checked off. Any system that handles the sentences somebody thought of, and fails on the rest, will feel unreliable no matter how well the thought-of cases work. Handling this properly needs measurement, not more examples.

1.3. Project as a solution

Remember Me was built to solve these problems in the order they are stated above. The user speaks or types in ordinary language, the application works out what the sentence means and what matters in it, and every later action finding, reminding, deleting, answering is performed by the application itself.

The key features of the application are the following:

- The user can store anything by saying or typing it in their own words. The application works out what kind of thing it is and what matters in it, rather than keeping it as a block of text.
- The user can ask for anything back as a question, and gets an answer in a sentence rather than a list of matching notes to read through.
- Voice input is converted to text on the device itself, so the recording never leaves the phone. Only the resulting text is sent.
- The user can ask the application to forget something, in words, and it is deleted.
- Reminders are set from the sentence itself. A stated deadline produces repeated reminders that get closer together as the deadline approaches; a stated moment produces a single reminder at that moment.
- Repeating reminders are stored as a rule rather than as a list of alarms, so a yearly reminder can be kept for years without scheduling thousands of notifications.
- Reminders can be answered back in words. Saying it is done stops them; saying it will not be done also stops them, and the difference between the two is recorded.
- Information the application judges to be sensitive, anything credential-like or medical is hidden on screen behind a tap, and left out of the browsing screens entirely.

- A photograph or document can be sent in the same conversation instead of a sentence, and retrieved at any time.
- Money can be recorded in the same conversation. Spending, income and transfers between the user's own accounts are tracked, and balances can be asked about in words.
- Two home screen widgets put today's remaining items, and a keypad for logging an expense, on the phone's home screen without opening the application at all.

1.4. Aims and objectives

1.4.1. Aims

This project aims to deliver a mobile application that lets a person store anything worth remembering by simply saying it, and get it back later by simply asking, so that the effort of remembering moves off the person and onto the application.

1.4.2. Objectives

i) To develop a mobile application for iOS and Android where a memory can be stored in ordinary spoken or written language, without the user having to choose a category, a folder or a format.

ii) To understand a sentence reliably enough that the stored result can be trusted, including when the sentence arrives as an untidy voice transcript.

iii) To answer a question from stored memories in a natural sentence, using only the information the user actually saved.

iv) To keep all finding, matching and filtering inside the application's own backend, so that the way a sentence is understood can be improved later without the behaviour of the application changing.

v) To set reminders from the meaning of the sentence, so that a deadline produces persistence and a fixed time produces a single alert.

vi) To store repeating reminders as a rule and schedule only the next occurrence, so that a lifelong reminder costs the same as a single one.

vii) To derive reminders the user did not explicitly ask for, from dated facts they stored, with enough lead time for the thing to actually be done.

viii) To protect sensitive memories on screen, and to keep voice recordings on the device.

ix) To build the whole system so that it can be run at low cost, and can grow without being rewritten.

1.5. Structure of the report

1.5.1. Background

This portion of the document will discuss the types of end user expected to use this application, and what they are expected to want from it. The technologies chosen for the project will be explained along with the reason each was chosen. The discussion on the topic of this application not being the only application of its type will take place, along with a comparison of similar applications.

1.5.2. Development

In the development section, a detailed walkthrough of the development process will be explained. It will outline the methodology used and the methodologies that were considered and rejected, the requirements the application was built against, and the design of the system, including the architecture, the data model and the flow of the main operations.

1.5.3. The application in use

This section will walk through the finished application screen by screen, using screenshots taken from the running application. It is included so that a reader who has not seen the application can follow the rest of the report.

1.5.4. Conclusion

Overall development and progress of the project will be summarised in the conclusion. This section will also reflect the learning, the challenges faced, and the objectives achieved throughout the development of the project. Additionally, the issues that may be faced socially, ethically or legally will also be discussed in this portion, followed by the advantages of the application.

Chapter 2. Background

2.1. Targeted Users

The application is aimed at ordinary people who have more to remember than they can comfortably hold, which is most adults. It does not assume any technical knowledge, and it deliberately does not ask the user to organise anything. There are no folders to create and no tags to apply, because a person who is willing to organise their information already has a system that works for them.

Within that, three groups were kept in mind during development:

i) People who carry a lot of small responsibilities for other people as well as themselves, a parent holding a family's appointments, documents and phone numbers. This group is the reason the application tracks who a stored fact is about, rather than assuming everything belongs to the user.

ii) People who find typing awkward or slow, including older users. This group is the reason voice input exists, and the reason the understanding is tested against untidy speech rather than tidy sentences.

iii) People who already keep this information somewhere but cannot get it back quickly. This group does not need to be convinced to store things. They need retrieval to work.

2.2. End User

The end user is a single individual using the application for themselves on their own phone. There is no shared account, no team, and no administrator role. Every memory belongs to exactly one user and is only ever visible to that user.

This was a deliberate decision rather than a simplification made to save time. Adding sharing to an application that holds passwords, medical information and account numbers changes what a single mistake costs. Until the application is trusted with the single user case, there is no case for widening it.

2.2.1. How requirements were gathered

The requirements for this project came from own use of the application during development, and from a running list of cases where it behaved wrongly. The application was installed on a real phone early and used daily, which is where most of the requirements in Chapter 3 actually came from. A number of the features described in this report, the deadline behaviour, the handling of the word "today", and the tense of an answer exist because the application got them wrong in normal use and the behaviour had to be corrected.

2.3. Understanding the solution

The solution has three parts. A mobile application the user talks to, a backend that holds the memories and does all of the finding, and an understanding step that works out what a sentence means and what matters in it. How that step works is deliberately not described in this report. What matters for the rest of the design is what it is not allowed to do: it never sees the user's stored memories in order to decide which one is relevant. That decision is made by the backend against the database.

2.3.1. Technologies

Layer	Choice	Why this one
Mobile application	React Native with Expo, in TypeScript	One codebase produces both the iOS and the Android application. Expo removes most of the native build setup, and its over-the-air updates meant the application on the phone could be updated during development without a rebuild each time.
Backend	Node.js with Fastify, in TypeScript	The same language on both sides, so types describing an API response are written once. Fastify was chosen over Express for its built-in schema validation and because it is measurably faster per request.
Database access	Prisma ORM	Gives typed database access, and generates migrations from the schema file, so the shape of the data and the code that reads it cannot drift apart silently.
Cloud database	PostgreSQL	Memories have a fixed part and a free-form part. The fixed part (owner, reminder time, deadline) is relational and gets indexed for fast reminder queries; the free-form part is stored in a JSONB column so a memory can hold any set of fields without a migration.

On-device database	SQLite through expo-sqlite	Memories the user chooses to keep local are stored only on the phone. SQLite is already present on both platforms and needs no server.
Voice to text	expo-speech-recognition, on device	The recognition runs on the phone using the platform's own engine, so audio never leaves the device. Only the resulting transcript is sent to the backend.
Notifications	expo-notifications, scheduled locally	Reminders are scheduled on the device itself, which means they still fire when the phone is offline and cost nothing to send.
Authentication	Custom JWT with bcrypt	A signed token the mobile application stores and sends with each request. Custom rather than a hosted identity provider so that no third party holds the user list.

Table 1: Technologies used, and the reason for each.

2.4. Similar Projects

2.4.1. Apple Reminders and Apple Notes

These are already on every iPhone, which makes them the real competition for anything in this space. Reminders accepts a spoken reminder through Siri and handles a fixed time well. Notes accepts any text at all. Between them they cover storing, and neither covers asking. A user cannot ask Notes what their wifi password is and be told; they can search for the word and be shown notes containing it, which is a different thing. Reminders also treats every reminder as an alarm at a moment, so a deadline has to be manually turned into a time by the user.

2.4.2. Google Keep

Keep is a fast, well-made notes application with reminders attached, and it syncs everywhere. Its weakness for this problem is the same one: it is organised around notes as objects the user manages. The user creates the note, labels it, colours it and later finds it. The work of organising is left with the person, which is exactly the work this project is trying to remove.

2.4.3. Todoist and similar task managers

Todoist has genuinely good natural language input typing "pay rent every 1st" creates a repeating task correctly. It is included here because its handling of recurrence was a direct influence. But it is a task manager, which means everything put into it has to be a task. There is nowhere in Todoist to put the fact that a friend's

mother's phone number is a particular string of digits, because that is not something to be done.

2.4.4. Mem and other AI note applications

A number of newer applications put a language model on top of a notes database and let the user ask questions of their notes. This is the closest existing category to this project. The differences are that they are built around long documents rather than single facts, they are desktop-first rather than voice-first, and they are paid subscriptions.

2.5. Comparison of similar projects

The applications discussed above more or less serve a similar purpose. However, some features are missing in those applications that a user of this kind of application would frequently want.

Feature	Remember Me	Apple Reminders / Notes	Google Keep	Todoist	Mem
Store anything in ordinary words	Yes	Yes	Yes	Tasks only	Yes
Ask a question and get an answer	Yes	No, search only	No, search only	No	Yes
Voice input processed on the device	Yes	Yes	Partly	No	No
Deadline produces repeated reminders	Yes	No	No	No	No
Repeating reminders stored as a rule	Yes	Yes	Yes	Yes	No
Reminders derived from stored facts	Yes	No	No	No	No
Answer a reminder in words	Yes	No	No	No	No
Sensitive memories hidden on screen	Yes	No	No	No	No
Money tracked in the same conversation	Yes	No	No	No	No
Free to use	Yes	Yes	Yes	Limited	No

Table 2: Comparison of Remember Me with similar applications.

2.6. Comparison analysis

The comparison shows that the gap is not in storing. Every application listed can store a piece of text, and the ones built into the phone can do it by voice as well. The gap is in everything that happens after storing.

Only the newest and most expensive of the alternatives can answer a question rather than return a search result. None of them treats a deadline differently from a fixed time, which is the single most common thing a person actually means when they say they need to do something. None of them creates a reminder from a fact the user stored without asking for one, and none of them hides a stored password from someone glancing at the screen.

Where inspiration was taken, it was taken openly. The recurrence handling was influenced by Todoist, and the idea that a question should be answerable at all was influenced by the newer AI note applications. The parts that are not found in any of the compared applications are the deadline behaviour, the derived reminders, and the spoken resolution of a reminder.

Chapter 3. Development

3.1. Considered Methodologies

I) Waterfall methodology. In this methodology the stages have to be well planned in advance, as there is no returning to a previous stage after moving on to the next. It was rejected because the requirements for this project were not knowable in advance. Several of the most important requirements only appeared after the application was in daily use.

II) Kanban methodology. Kanban suits maintenance work and ongoing projects where tasks arrive steadily. It was rejected because it provides no time constraint, and this project needed one to stop the language work expanding without end.

III) Spiral methodology. The spiral model handles risk well through repeated cycles of prototyping and evaluation, which suits the uncertain parts of this project. It was rejected as too heavy for a single developer, since much of its structure exists to manage risk across a team.

(Further detail on each of the considered methodologies can be found in Appendix C.)

3.2. Selected Methodology

Agile methodology was selected for this project, worked in short iterations. Each iteration took one feature from the requirement list, built it, put it in front of a real user on a real device, and only then merged and deployed it.

Agile was the correct choice here for one specific reason rather than a general preference. This application cannot be judged from a specification. Whether the reminder behaviour is right, or whether an answer reads correctly, is only visible when a real person says a real sentence to it. Several features in this report were built, tried, found wrong, and rebuilt within the same iteration, which no planned methodology would have allowed.

The working rule for every feature was fixed, and not skipped:

i) Create a branch off the main development branch before any change.

ii) Build the feature, verify the backend against a local database, and typecheck the mobile application.

iii) Put it in front of the user in the iOS Simulator, running against the local backend, so nothing needs to be deployed to try it.

iv) Wait for explicit approval. Building is not approval.

v) Only after approval, merge and push, which is what deploys the backend.

vi) Installing on a real phone is a separate, later step, done against the deployed backend.

(The phases of the methodology are described in Appendix A, and the reasoning behind selecting it in Appendix B.)

3.3. Work plan

The work was planned in blocks rather than on a dated chart, since the project ran continuously rather than to a fixed external deadline. The order below is dependency order each block needed the one above it to exist first.

Block	Work	Depends on
1	Project setup, database schema, authentication with email verification and password reset	—
2	Understanding a sentence, storing and reading back memories	Block 1
3	Voice capture on device, chat screen, saving and recalling from the application	Block 2
4	Reminders: scheduling, the pacing engine, answering a reminder in words	Block 2
5	Today screen, calendar browsing, saved memories, sensitive-memory masking	Blocks 3 and 4
6	Photographs and documents: sending one in, retrieving it later	Block 2
7	Money: accounts, transactions, transfers, budget screen, logging and asking in words	Blocks 2 and 5
8	Lifelong reminders: recurrence rules, derived reminders, ownership of a stored fact	Block 4
9	Search and matching done in the backend	Blocks 2 and 5
10	Home screen widgets	Blocks 5 and 7
11	Scale and security review, and hardening for growth	All of the above

Table 3: Indicative work plan, in dependency order.

3.4. Requirement analysis

The requirements were separated into functional requirements, which describe what the application must do, and non-functional requirements, which describe how well it must do it. Each was given a priority of high, medium or low. High means the application is not usable without it.

ID	Requirement	Priority
FR-01	A user can create an account with an email address and a password, and must verify the email address before logging in.	High
FR-02	A user can log in, stay logged in between launches, and log out.	High
FR-03	A user can reset a forgotten password through a code sent to their email address.	High
FR-04	A user can store a memory by typing it in ordinary language.	High
FR-05	A user can store a memory by speaking it, with the speech converted to text on the device.	High
FR-06	The application extracts the useful values from the sentence and stores them as named fields.	High
FR-07	A user can ask a question in ordinary language and receive an answer built from their stored memories.	High
FR-08	A user can ask the application to forget a specific memory, and it is deleted.	High
FR-09	A user can ask the application to forget everything.	Medium
FR-10	The application sets a reminder when the sentence implies one, at the time the sentence implies.	High
FR-11	A stated deadline produces repeated reminders that get closer together as the deadline approaches.	High
FR-12	A repeating reminder is stored as a rule, with only the next occurrence scheduled.	High
FR-13	A user can report a reminder as done, or as not going to be done, in words.	High
FR-14	A user can change when a reminder is due, or how often it repeats, in words.	Medium
FR-15	The application can create a reminder from a dated fact the user stored without asking for a reminder.	Medium
FR-16	A memory records who it is about, so a document belonging to a family member is not treated as the user's own.	Medium
FR-17	Credential and medical memories are masked on screen and excluded from browsing screens.	High
FR-18	A user can store a photograph, have text read out of it, and have the image kept only when the image itself is the record.	Medium
FR-19	A user can see everything due today, grouped by part of the day, and mark items done.	High
FR-20	A user can browse reminders by date on a calendar.	Medium
FR-21	A user can create accounts and record expenses, income and transfers against them.	Medium
FR-22	A user can record a transaction and ask about a balance in ordinary language.	Medium

FR-23	A user can see today's remaining items and log an expense from a home screen widget.	Low
FR-24	A user can delete their account.	High

Table 4: Functional requirements.

ID	Requirement	Priority
NFR-01	A memory is only ever visible to the user who created it, enforced on the server for every request.	High
NFR-02	Passwords are stored only as a hash, never in a form that can be read back.	High
NFR-03	Voice recordings never leave the device.	High
NFR-04	The application responds to a typed sentence in a time that feels immediate; understanding a sentence should stay near one second.	High
NFR-05	Answering a question must not require sending the user's whole memory store anywhere.	High
NFR-06	Repeated login attempts against one account are limited regardless of where they come from.	High
NFR-07	Every external call has a timeout, so one slow third-party service cannot hold requests open.	High
NFR-08	The reminder sweep must be safe to run with more than one server instance, without sending a reminder twice.	High
NFR-09	The language understanding must be measured against labelled cases, not judged by hand.	High
NFR-10	The application must cost nothing to keep running.	Medium

Table 5: Non-functional requirements.

(The full requirement analysis, including the use case, can be found in Appendix D.)

3.5. Design

This section describes the design of the system. There are no diagram images in this report, so the architecture, the data model and the main flows are written out in words instead, in enough detail that a diagram could be drawn from the description, or the same behaviour rebuilt in a different language.

3.5.1. System architecture

The system is made of four parts that run in four different places.

i) The mobile application, running on the user's phone. It holds the screens, the voice capture, the on-device database for local-only memories, and the scheduling of notifications. It is the only part the user ever sees.

ii) The backend API, running as a single service. It holds the accounts, the cloud memories, the money records, and all of the logic that decides which memory answers a question.

iii) The database, PostgreSQL, running as a managed instance. Only the backend talks to it. During development it runs as a container on the developer's machine instead.

iv) The external services: an email service for verification and password reset emails, and the phone platform's notification service. Both are reached only from the backend, never from the phone, so no key for either is ever shipped inside the application.

The flow of a request is always the same shape. The phone sends an HTTPS request carrying a signed token. The backend checks the token, loads the user, and refuses the request if the token was issued before the last password reset. It then does the work, talking to the database, and returns the result, and the phone updates what is on screen.

There is one part that runs without any request at all. A scheduled job inside the backend wakes every minute, looks for cloud reminders that are due and have not been sent, and sends them. Because more than one copy of the backend may eventually run at once, that job takes a database advisory lock before doing anything, so only one copy sweeps at a time and a reminder cannot be sent twice.

3.5.2. The two places a memory can live

A memory is either a cloud memory or a local-only memory, and the difference is not a synchronisation setting. A local-only memory is never stored on the backend at all. It lives in the SQLite database on the phone, holding only the extracted category and fields, and not even the original sentence.

The fields of a local memory do briefly travel to the backend during recall, because the wording of an answer is produced there. What never travels is the store as a whole: the phone narrows its own memories down first and sends only the handful that could possibly answer the question.

3.5.3. Logical data model

There are five tables. The design decision that matters most is in the first one.

Entity	What it is	Notable columns
User	One person's account.	email, passwordHash, emailVerified, failedLoginAttempts, lockedUntil, tokenVersion, isActive, name, preferredName, dateOfBirth, homeCurrency, pushToken
Memory	One thing the user asked to be remembered.	category, kind, fields (JSONB), remindAt, deadlineAt, recurrenceRule (JSONB), repeatEveryMinutes, reminderState, reminderPhrase, completedAt, leadTimeDays, origin, parentMemoryId, subject, subjectIsSelf, imageKey, isActive
Account	One place the user keeps money.	name, currency, kind, openingBalanceMinor, isPrimary, payFromAccountId, archived
TxCategory	A spending or income category.	name, icon, colorKey, flow, isSeeded, archived
Transaction	One movement of money.	type, amountMinor, currency, title, occurredAt, accountId, categoryId, transferAccountId, transferAmountMinor

Table 6: The main entities and what they hold.

A user has many memories, many accounts, many categories and many transactions, and every one of those rows carries the user's id. A transaction belongs to exactly one account and optionally to one category. A transfer is a single transaction that names a second account. A memory can have a parent memory, which is how a derived reminder is linked to the fact it came from; deleting the parent deletes its derived children with it.

Why category and kind are two different columns

It looks like duplication and it is not. The category is free text written by the application for the user to read "travel", "gift", "boiler" and nothing in the code ever branches on it. Even the colour of a memory card is a hash of the string rather than a lookup, so a category that has never been seen before still gets a colour.

The kind is a closed set that the code does branch on: place, credential, medical, or other. Two of those, credential and medical, decide whether the memory is hidden on screen.

They were one column to begin with, and that produced a real bug: a memory whose category happened to contain the word "workplace" was matched as a place and

shown as somewhere to go. Splitting the column fixed it. The kind is deliberately not a database enum but a plain string validated in code, so that adding a new kind later costs no migration.

Why amounts are stored as integers

Every amount of money in the database is stored in minor units cents as an integer, never as a decimal number. Storing money as a floating point number means that adding a series of ordinary prices eventually produces a total that is a fraction of a cent wrong, and in a budget that is visible to the user. Integers cannot drift.

3.5.4. Flow: storing a memory

This is the flow that runs when a user says or types something to be remembered. It is written out step by step because everything else in the application depends on it.

i) The user is on the chat screen. They either type into the input box or hold the microphone button. If they speak, the phone's own speech recognition converts the audio to text on the device; the audio is never uploaded.

ii) The application sends the text to the backend, along with the phone's current local time and timezone offset, the last few turns of the conversation, and the memory currently being discussed if there is one.

iii) The backend checks the token and identifies the user.

iv) The backend works out what the sentence means and what matters in it. Relative words like "today" and "tonight" are resolved into real dates at this point, before anything is stored, using the phone's clock.

v) The backend decides whether this is a reminder at a fixed moment or a deadline that should be chased, and how often to repeat.

vi) The result is sent back to the phone. Nothing has been stored yet.

vii) The phone stores the memory, either in the local SQLite database or by calling the backend to store it in Postgres, according to the user's choice.

viii) If the memory carries a reminder, the phone schedules the notifications locally. If it is a repeating reminder, only the next occurrence is scheduled.

ix) The phone shows a short confirmation of what was understood, so the user can see immediately if it was taken the wrong way.

The reason the memory is stored by the phone rather than by the backend during understanding is that the user has not yet agreed to anything at step seven. If the understanding needs a clarifying question, the memory must not already exist.

3.5.5. Flow: asking a question

This flow is the one that changed most during development, and the final design is the point of the whole project.

i) The user asks a question in the chat screen.

ii) The backend works out what the question is asking for: a time window if the question had one, and the thing it named if it named something. Words like "today" are treated as a date, not as something to match text against.

iii) The application sends that to the backend's search, or runs it against the on-device database for local memories.

iv) The backend finds the matching memories in the database. A named thing is matched against what was stored; a time window is matched against the reminder time or the deadline. A question that names something is not filtered by date, because the date of a named thing may only exist inside its text.

v) The handful of matching memories never more than ten are used to write one natural sentence answering the question.

vi) Before that sentence is written, the backend works out whether each matching memory is in the past, is today, or is still ahead, by comparing its date to the user's clock, and that verdict is used in the wording.

vii) The sentence is returned and shown in the chat.

Step six exists because whether a date has passed is a comparison of two dates, which is ordinary code, so it is done in ordinary code and the answer is handed over as a fact. Without it, a memory dated in August and asked about in September would be answered in the future tense.

3.5.6. Flow: reminders and how often they repeat

The application distinguishes between two things that other applications treat as one.

A fixed reminder is a single notification at a stated moment. "Remind me at six to take the chicken out" is a fixed reminder. It fires once.

A deadline-driven reminder is different. "The rent has to be paid by today" states a deadline, not a time, and what the user wants is to be told about it repeatedly until it is done. The application schedules a series of reminders whose spacing depends on how much time is left: far from the deadline they are far apart, and as the deadline approaches they get closer together. The escalation is not a special case that had to be written; it falls out of making the gap a function of the time remaining.

The pacing engine that does this is ordinary, deterministic code on the device. The only judgement involved is whether the thing is important, which changes the default spacing when no deadline was given.

A repeating calendar reminder every Monday, every year on the 14th of September is stored as a rule, and only the next occurrence is ever scheduled. A phone will only hold a limited number of pending notifications, so scheduling a yearly reminder for the next fifty years is not possible even if it were sensible. When the reminder fires, or when the application is next opened, the rule is used to work out the next occurrence and that one is scheduled.

Re-arming on opening the application matters more than it sounds. A notification the user never taps produces no event, so an application that only reacts to taps would quietly stop repeating. Instead, on opening, the application compares what should be scheduled against what actually is, and fixes the difference.

3.5.7. Flow: reminders the user never asked for

Some facts imply an action even though the user only stated the fact. A photographed driving licence has an expiry date on it. A stored birthday comes round every year.

When a memory carries a dated fact of this kind, the application offers reminders derived from it. Two things make this useful rather than annoying. The first is lead

time: the reminder starts far enough ahead that the thing can actually be done, so a document that takes a week to renew is raised a month before it expires rather than on the day. The second is ownership. If the stored document belongs to a family member rather than the user, a renewal reminder is not created, because renewing it is not the user's job. A birthday is treated the opposite way wishing somebody a happy birthday is the user's action regardless of whose birthday it is.

Working out whether the document belongs to the user is done by comparing the name on it to the account holder's first name only. Matching on the surname was tried first and was wrong: a brother's passport shares the surname and is not the user's document.

Chapter 4. The application in use

This chapter walks through the finished application using screenshots taken from it running on an iPhone. Every screenshot in this chapter is of the real application with real stored data, not a mockup. The data shown belongs to a demonstration account.

There are four tabs along the bottom, Home, Today, Budget and Profile, and a Chat button that floats above all of them. Chat is where everything is said to the application; the tabs are ways of looking at what it has understood. Notes, Clock, Calendar and Saved open from Home or from Profile rather than taking a tab of their own.

4.1. Home

Home is the screen the application opens on. It answers the question a person actually has when they pick up their phone, what is waiting for me, before they have to go looking for it.

Two counts sit at the top: what is due today, with anything already past its time called out as overdue, and what has been finished. Below them are shortcuts into the four things people open most, and under that the memories most recently stored, each shown as a card carrying its kind and its named values.

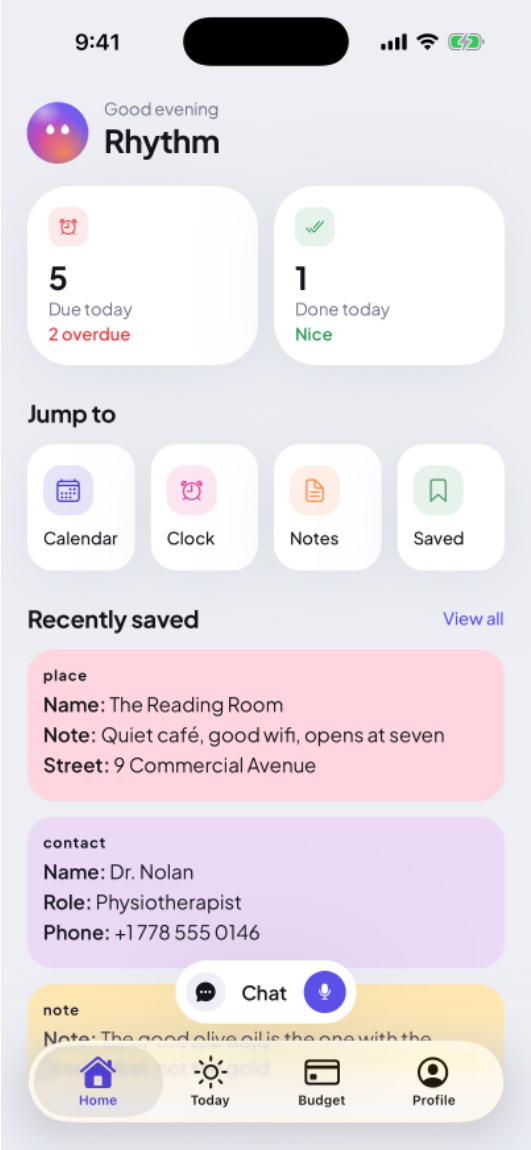

Figure 1: Home: what is due, what is done, and what was just stored.

The colour of a card comes from the category text the application wrote for it, so memories of the same sort look alike without any list of categories existing in the code.

4.2. Today

Today takes the same information and lays the day out in order. At the top is a count of what is left with a ring showing progress through it, then a rail of the five parts of the day, from sunrise to night, each carrying the number of items in it.

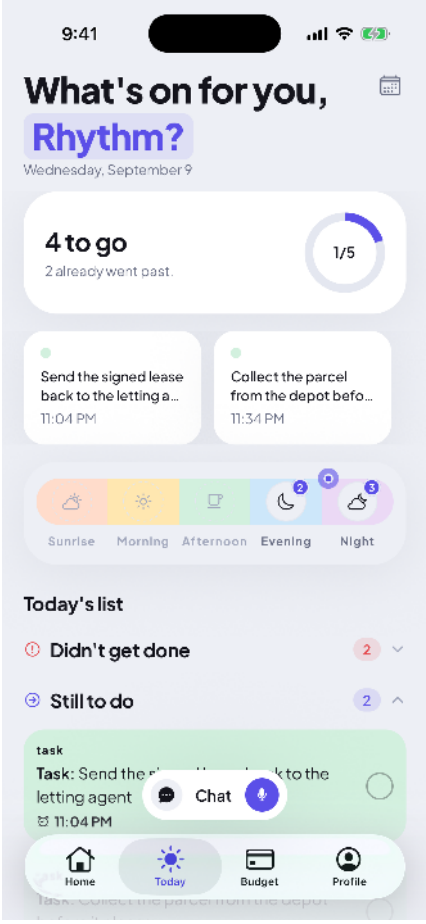

Figure 2: Today, with the day rail and what is left.

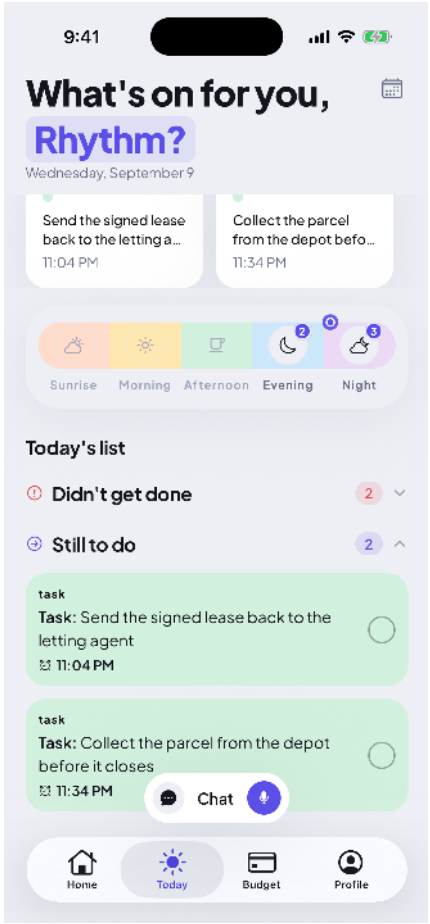

Figure 3: Items that were not done are kept, not dropped.

The reason for the day rail rather than a single list is that a list of eleven things is a list nobody reads. Splitting the day into parts means the question the screen answers is "what is left this evening", which is a question with a short answer. Anything whose time has passed is marked overdue rather than hidden, and anything not done by the end of its day is collected into a section of its own rather than quietly disappearing.

The calendar icon at the top opens a month view, where any day can be selected and its reminders read. This is the way to look forward or back rather than at today. A dot under a date means something is on it.

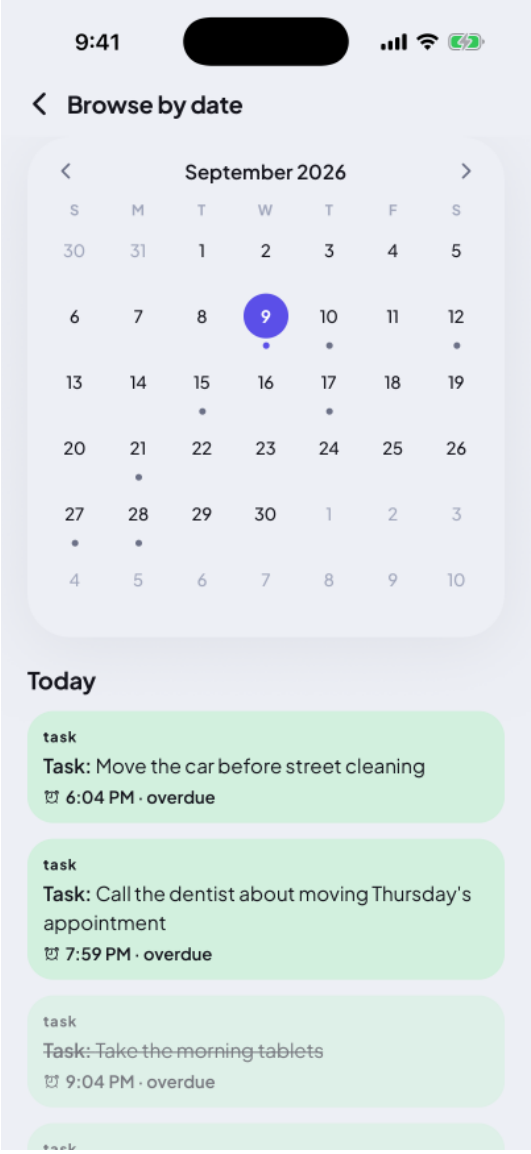

Figure 4: Browsing reminders by date.

4.3. Chat

Chat is the whole of the input. There are no forms anywhere in the application for creating a memory, choosing a category or setting a reminder time. Everything is said in a sentence. The three controls at the bottom are the microphone, the image picker and the text box; the microphone converts speech to text on the device itself.

4.3.1. Storing something

The user says or types the thing in their own words. The application confirms what it understood, in a short sentence, so a misunderstanding is visible immediately rather than a week later when the information is needed.

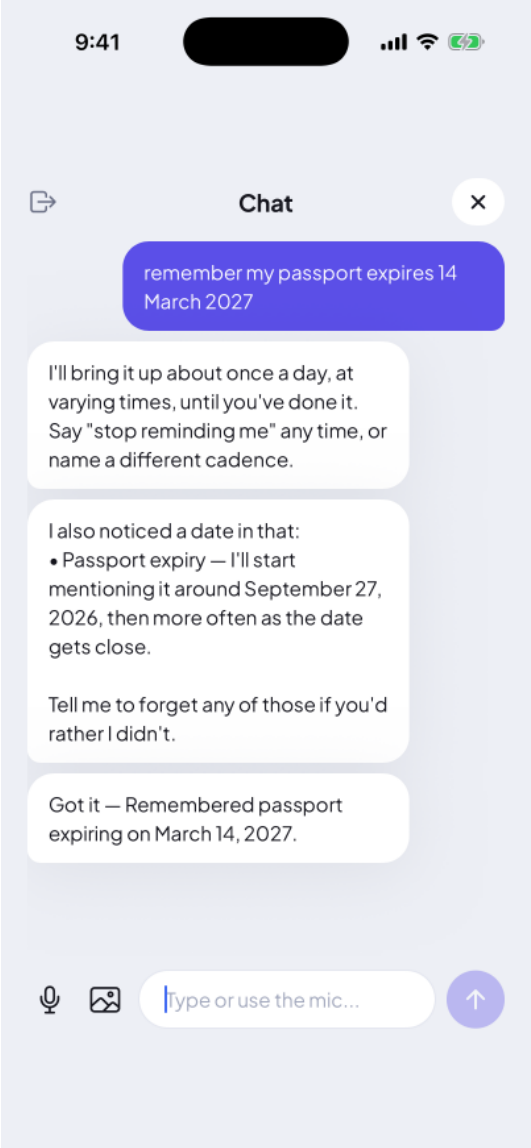

Figure 5: Storing a memory by typing it in ordinary words.

Two things happened here that were never asked for. The application noticed the sentence carried a date worth acting on and offered to start mentioning it well before the passport actually expires, and it said plainly how often it intends to bring it up and how to make it stop. What was stored is not the sentence either it is a memory with a named expiry date in it, which is what makes the next screenshot possible.

4.3.2. Asking for it back

A question is asked the same way. The application finds the memory in its own database, and the answer is a sentence containing the stored value.

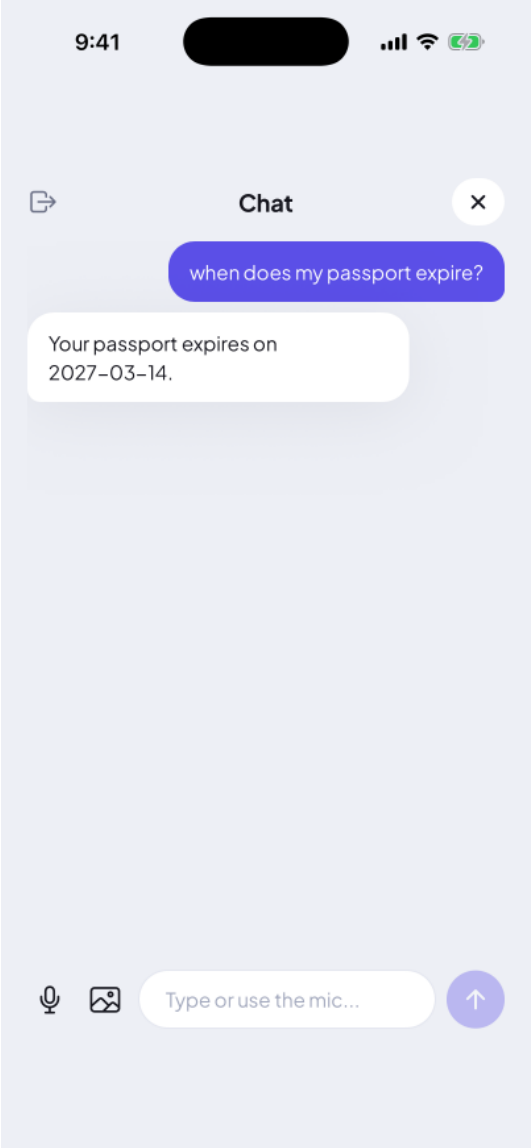

Figure 6: Asking a question and being told the answer.

This is the difference from a notes application. The user did not search for the word "passport" and read a note. They asked a question and were told the answer.

4.3.3. Deciding how hard to push

When a sentence carries a deadline rather than a fixed time, the application asks how it should behave about it, because those are genuinely two different things. One nudge on the day is right for an appointment. A deadline that has to be met needs to be brought up more than once.

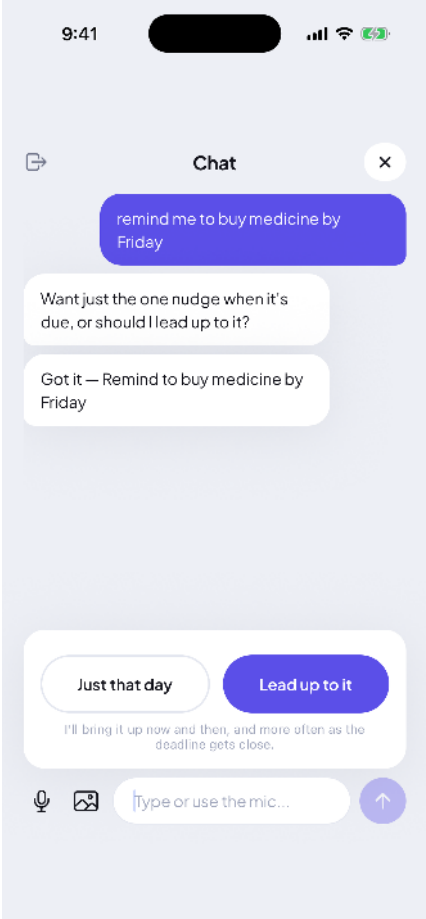

Figure 7: One nudge, or lead up to it the user decides.

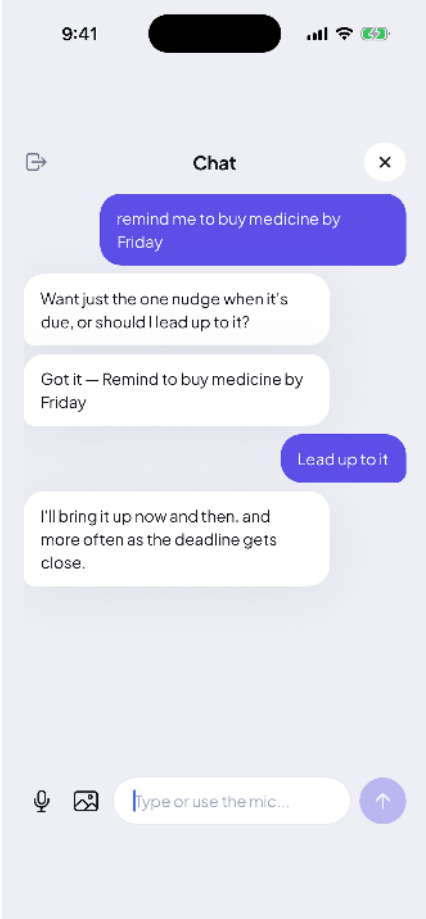

Figure 8: Having chosen, the application says what it will actually do.

This is where the application's judgement of priority matters most. Judging that buying medicine by Friday deserves persistence, while a note about a café does not, is what separates an application that helps from one that either nags about everything or stays silent about the thing that mattered.

4.4. Notes

Anything stored can also be written out longhand. Notes holds documents that can be edited directly, arranged into folders, alongside the memories that came in through conversation.

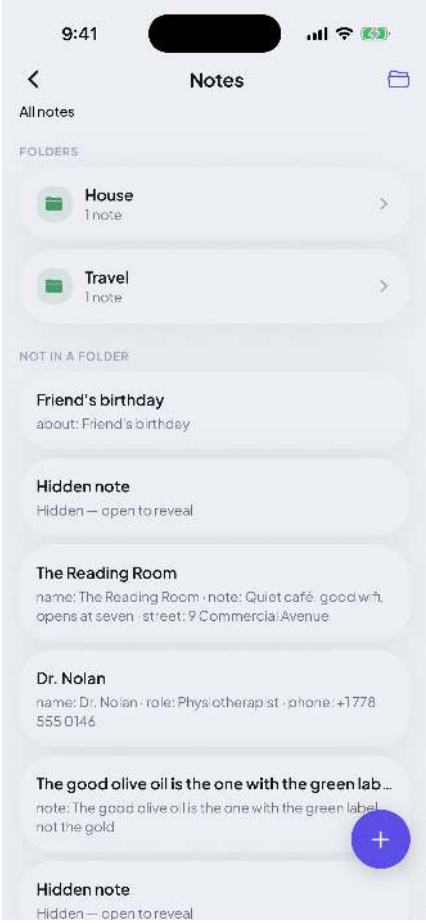

Figure 9: Notes, with folders and everything not in one.

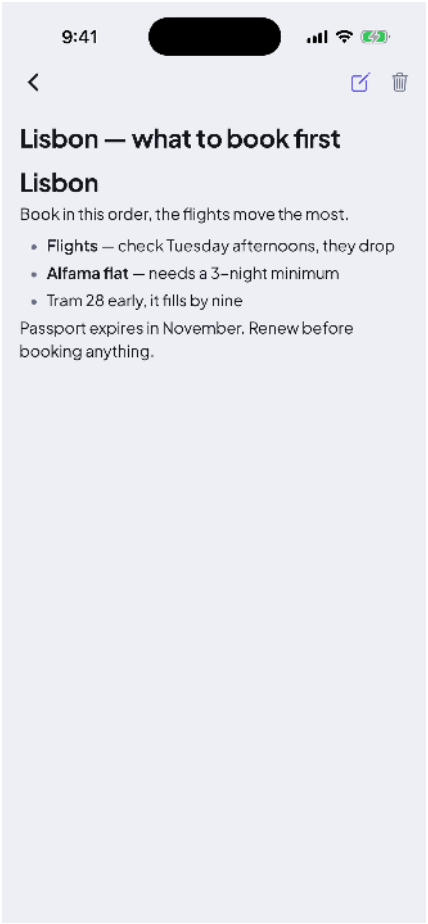

Figure 10: A note opened for editing.

4.5. Clock

The application includes its own alarms. They are stored on the phone, so they work with no connection at all and follow the user across timezones, and they are real system alarms they ring through silent and through Focus, which an ordinary notification does not.

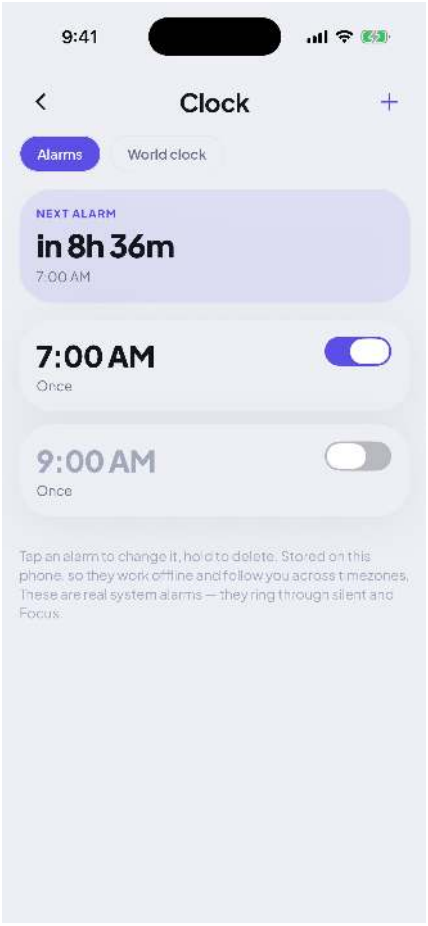

Figure 11: Alarms, with the next one counted down.

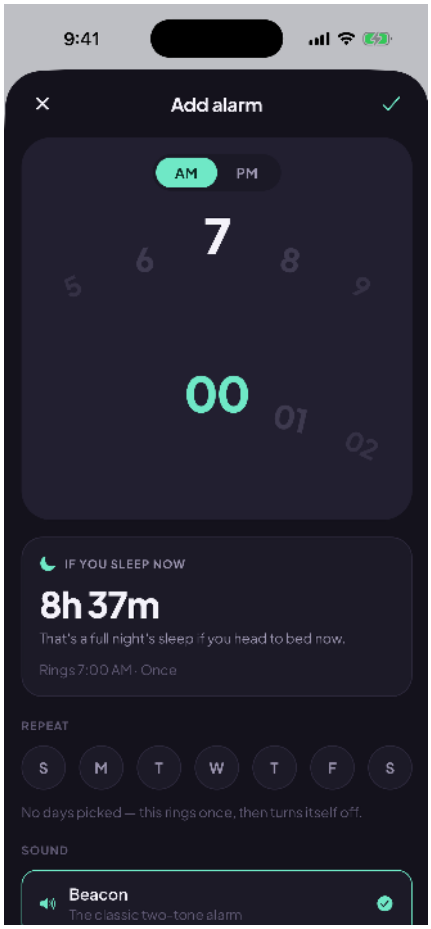

Figure 12: Setting an alarm, with the sleep it leaves worked out.

Setting one shows how much sleep it actually leaves if the user goes to bed now, which is the number a person is really trying to work out when they set a morning alarm.

4.6. Saved memories

Everything stored can be browsed in one place. Each memory is shown as a card carrying its kind, its named values, the date it was stored, and what the application intends to do about it.

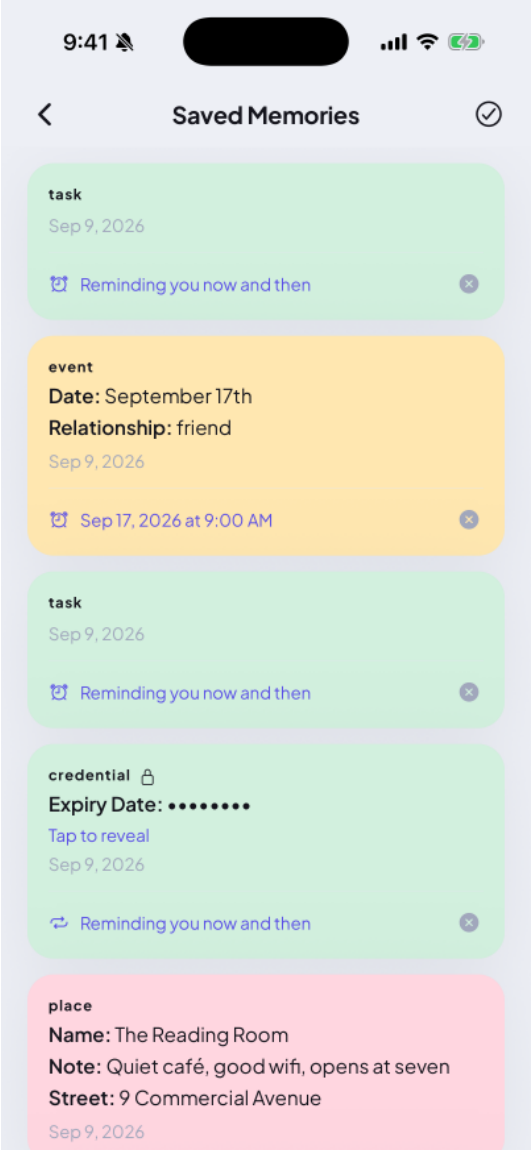

Figure 13: Saved memories, showing the extracted values rather than the original sentences.

Memories the application judged to be sensitive are masked. A medical note or a credential shows its field names but not its values, with a lock icon and a line inviting a tap to reveal.

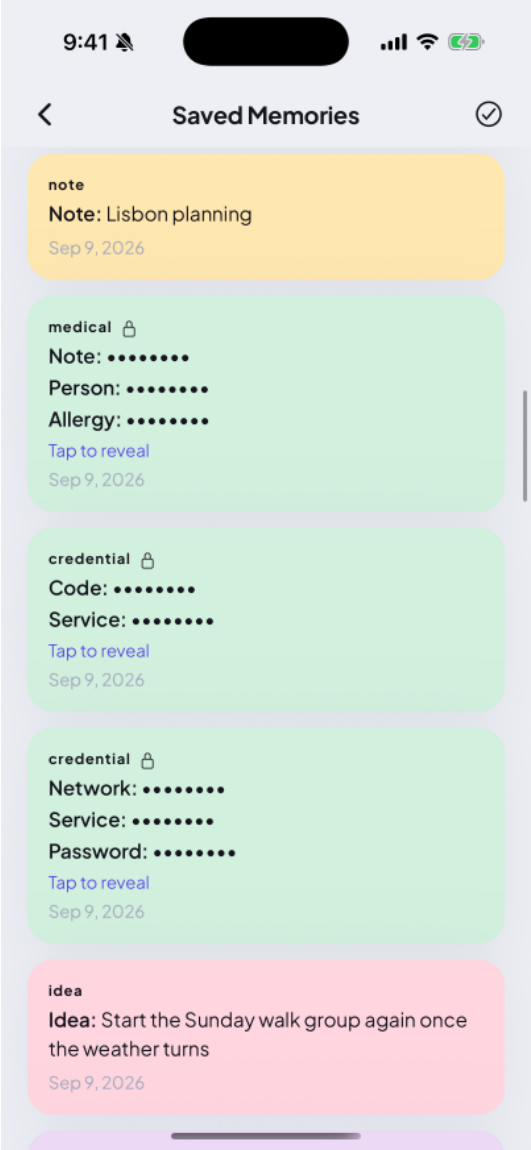

Figure 14: Sensitive memories are masked until tapped.

This exists because the most useful things to store are the ones most damaging to show. A password that is only safe as long as nobody glances at the phone is not really stored safely at all.

The judgement will sometimes be wrong, so it can be overridden. Holding a memory offers a set of actions, which include marking it as sensitive, correcting what the application decided it was, and renaming its label. The application decides by default and the user has the last word. Activities, reached from Profile, groups the same memories by what they are; sensitive kinds are left out of it entirely rather than shown masked, because a screen designed for browsing is exactly where a credential should not appear at all.

4.7. Budget

The money side of the application uses the same conversation. An expense can be logged by saying it, and a balance can be asked for in words, but there is also a full screen for looking at it: net worth across every account, the accounts themselves, spending over the last fortnight, and what happened recently.

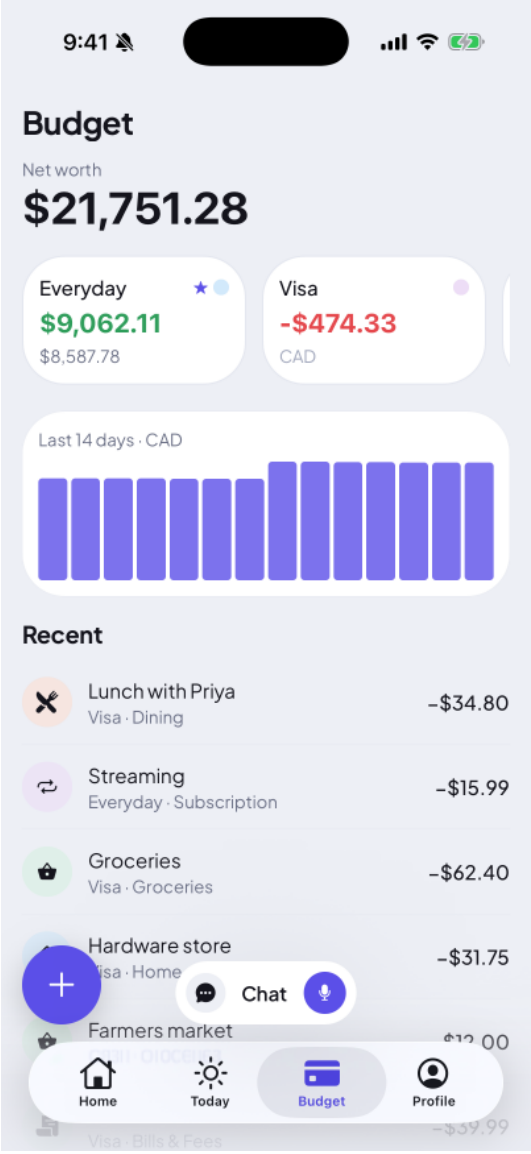

Figure 15: The budget screen: net worth, accounts, recent activity.

Each account has its own history. Money moved between the user's own accounts is stored as a single transaction naming both sides rather than as two separate records, so a transfer can never be half recorded.

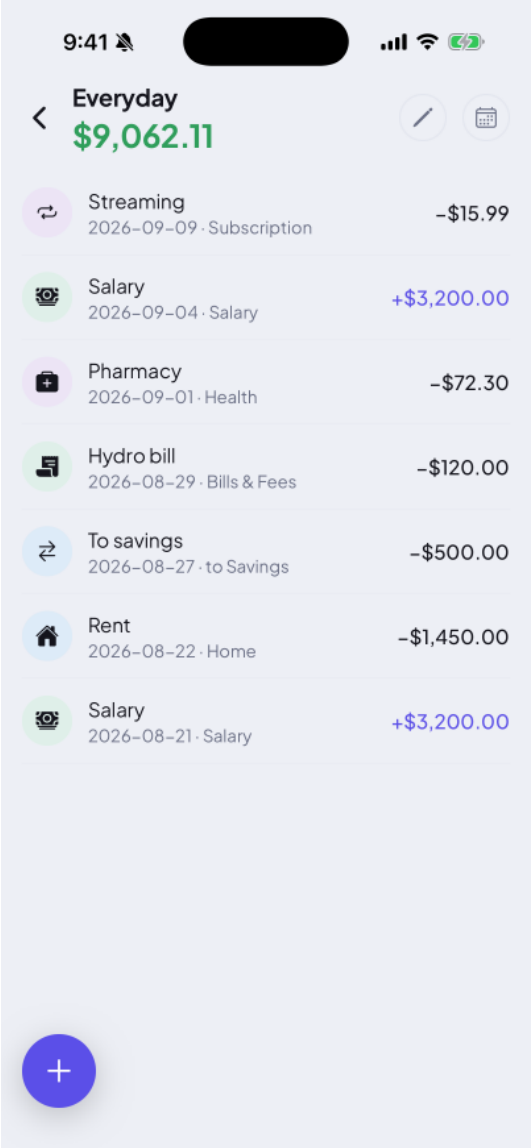

Figure 16: One account's transaction history.

The rule the application follows for money in conversation is deliberately strict. Saying "spent 40 on groceries from my credit card" records a transaction, because an account was named. Saying "spent 40 on groceries" does not, because the application cannot know which account it came from, and a wrong entry in a budget is worse than a missing one. That sentence is stored as an ordinary memory instead.

4.8. Profile

The profile screen holds what the application should call the user, their date of birth if they want birthday handling, the way into Notes, Clock, Saved and Activities, the appearance setting, the privacy policy and terms, and the two account actions.

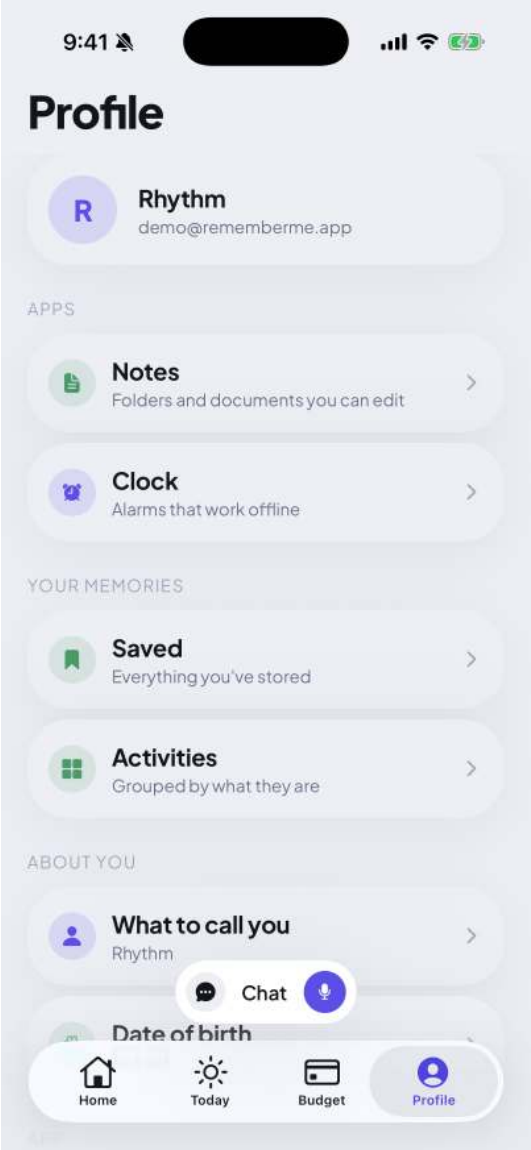

Figure 17: The profile screen.

Deleting the account is offered directly on this screen rather than buried, since an application holding this kind of information should make leaving easy. The application follows the phone's light or dark setting by default and can be forced either way; every screen is built from the same set of colour tokens, so a screen cannot be light in a dark application by accident.

4.9. Home screen widgets

Two widgets put the application on the phone's home screen. The first shows what is left today and updates as items are completed, marking anything overdue. The second is a keypad for logging an expense without opening the application at all.

Both read and write the same data as the application itself, so an expense entered on the home screen appears in the budget immediately.

Figure 18: The Today widget on the phone's home screen.

Chapter 5. Conclusion

The project set out to build an application where a person can store anything by saying it and get it back by asking. That application exists, runs on a real phone, and is used daily. The objectives set out in Chapter 1 have been met.

The most valuable thing learned during the project was not technical. It was that fixing reported faults one at a time feels like progress and is not. Judging the application by trying a few sentences by hand will always flatter it, because the

sentences that get tried are the ones already known to work. Real confidence came only from checking it against phrasings nobody had thought of in advance.

5.1. Legal, Social and Ethical Issues

5.1.1. Legal issues

This application is operated from Canada, so Canadian law applies to it. The main concern is that the application deliberately collects exactly the information that privacy law treats most seriously: passwords, account numbers, identity document numbers, and health information such as prescriptions and allergies.

Under the Personal Information Protection and Electronic Documents Act (PIPEDA), an organisation that collects personal information in the course of commercial activity must obtain meaningful consent, collect only what it needs for a stated purpose, protect it with safeguards appropriate to its sensitivity, and let a person access and correct what is held about them (Office of the Privacy Commissioner of Canada, 2019). The Act specifically expects the level of protection to rise with the sensitivity of the information, which is directly relevant here, since a memory store of this kind is more sensitive than most databases of its size.

PIPEDA also requires that a breach creating a real risk of significant harm be reported to the Privacy Commissioner and to the people affected, and that a record of every breach be kept whether or not it was reportable (Government of Canada, 2018). For an application holding credentials, almost any breach would meet that threshold.

Two further pieces of Canadian law apply. Canada's Anti-Spam Legislation governs commercial electronic messages, which affects any future messaging beyond the verification and password reset emails the application already sends. And in Quebec, Law 25 imposes stricter requirements than PIPEDA, including a named privacy officer and explicit consent for each purpose, which would apply to any user there.

The application's design already answers part of this. Voice is converted to text on the device, so no recording is transmitted or stored anywhere. Passwords are stored

only as hashes. A user can delete any memory, or their whole account, from within the application.

5.1.2. Social Issues

The obvious social concern with an application like this is dependence. An application that remembers things for a person may leave that person less able to remember things themselves. This is a real effect and it is not unique to this application it is the same argument that was made about written phone numbers and about calculators but it is worth stating rather than dismissing, because the application is designed to be relied on.

A second concern is that the application currently only works properly in English. A person who thinks in another language gets a worse application. For a tool whose whole purpose is understanding how somebody naturally speaks, that is a real limitation rather than a missing feature.

A third concern is who this kind of application is actually for. The people who would benefit most from something that remembers on their behalf include older users and people with memory difficulties, and those are the same people least likely to trust an application with a prescription or a bank detail. Earning that trust is a design problem, not a marketing one. (Additional detail can be found in Appendix F.)

5.1.3. Ethical issues

The central ethical issue is what the application decides to hide. It judges which memories are sensitive, and it will sometimes be wrong. Getting it wrong in one direction shows a password on a screen in public; getting it wrong in the other direction hides something harmless behind an extra tap. The application is deliberately biased towards hiding, because that mistake is recoverable in a way the alternative is not.

A second is honesty about how well it actually works. An application that people trust with a medication dose should not advertise a confidence it cannot defend, and should say plainly where it is known to be weak with languages other than English, and with sentences phrased in ways it has not met before. (Additional detail can be found in Appendix F.)

5.2. Advantages

The Remember Me application comes with a number of benefits, some of which are not offered by any of the applications it was compared against:

i) Anything can be stored by simply saying it, with no category, folder or format for the user to choose.

ii) A question is answered with a sentence containing the stored value, rather than with a list of documents to read through.

iii) A stated deadline produces reminders that keep coming and get closer together as the deadline nears, which is what a person means when they say something has to be done by today.

iv) A repeating reminder costs the same as a single one, because it is stored as a rule rather than as a list of alarms.

v) Reminders can be answered in words, and the difference between having done something and having decided not to is recorded.

vi) Reminders can be created from facts the user stored without asking for one, with enough notice that the thing can actually be done.

vii) A stored document belonging to a family member is not treated as the user's own, which prevents a whole class of wrong reminders.

viii) Voice never leaves the device, which is a stronger privacy position than any of the compared applications offers.

ix) Sensitive memories are hidden from a glance at the screen and left out of browsing screens entirely.

x) Money and memory live in one conversation, so an expense can be logged in the same sentence style as anything else.

xi) The whole system costs nothing to keep running.

Chapter 6. References

Expo, 2026. *Expo SDK 54 documentation*. [Online] Available at: https://docs.expo.dev/

Fastify, 2026. *Fastify documentation*. [Online] Available at: https://fastify.dev/docs/latest/

Government of Canada, 2018. *Breach of Security Safeguards Regulations (SOR/2018-64)*. [Online] Available at: https://laws-lois.justice.gc.ca/eng/regulations/SOR-2018-64/

Government of Canada, 2000. *Personal Information Protection and Electronic Documents Act (S.C. 2000, c. 5)*. [Online] Available at: https://laws-lois.justice.gc.ca/eng/acts/P-8.6/

Office of the Privacy Commissioner of Canada, 2019. *PIPEDA fair information principles*. [Online] Available at: https://www.priv.gc.ca/en/privacy-topics/privacy-laws-in-canada/the-personal-information-protection-and-electronic-documents-act-pipeda/p_principle/

PostgreSQL Global Development Group, 2026. *PostgreSQL documentation: JSON types*. [Online] Available at: https://www.postgresql.org/docs/current/datatype-json.html

Prisma, 2026. *Prisma ORM documentation*. [Online] Available at: https://www.prisma.io/docs

Chapter 7. Appendix

7.1. Appendix A: Phases of the methodology

7.1.1. Initiation phase

The problem was stated and the shape of the solution decided: what the application would do, who for, and which parts of it were genuinely uncertain. The uncertain parts were listed deliberately, because those are the ones that decide whether an iterative methodology is needed at all.

7.1.2. Planning phase

At the start of each iteration one feature was chosen from the requirement list, broken into the changes needed on each side, and checked against what already existed so that it would not duplicate something built earlier.

7.1.3. Execution phase

The feature was built on its own branch. Backend changes were verified against a local database, and the mobile application was type checked. Nothing was merged during this phase.

7.1.4. Review phase

The feature was put in front of the user in the iOS Simulator, running against the local backend, so that it could be tried without being deployed. This is the phase that produced most of the corrections in this report, because the difference between a feature working and a feature being right is only visible here.

7.1.5. Retrospective phase

What went wrong was recorded rather than only fixed. The conclusions drawn in Chapter 5 come from this phase, including the realisation that fixing reported faults one at a time was not the same as knowing the application worked.

7.1.6. Release phase

Only after explicit approval was the branch merged and pushed, which deploys the backend automatically. Installing on a real phone was treated as a separate step afterwards.

7.2. Appendix B: Selected methodology

Agile was selected because the requirements for this project could not be known in advance, and because the quality of the result depended on a judgement whether an answer reads correctly, whether a reminder is annoying that cannot be written into a specification and can only be made by using the thing.

The concrete evidence for that choice is in this report. The deadline behaviour, the handling of the word "today", the tense of an answer, and the ownership rule for stored documents were all corrections made after using the application, not decisions made before building it. Under a planned methodology every one of those would have shipped wrong.

7.3. Appendix C: Considered methodologies

7.3.1. Waterfall methodology

Waterfall runs each stage once, in order, with no return to a previous stage. It suits projects whose requirements are fixed and well understood at the start, and where the cost of a change late on is high. It was rejected here because the requirements were not fixed: the most important ones only became visible after the application was in use.

7.3.2. Kanban methodology

Kanban visualises work as a flow and limits how much is in progress at once. It suits ongoing maintenance and support work well. It was rejected because it deliberately provides no time boxing, and this project needed a boundary around iterations to stop the language work expanding indefinitely.

7.3.3. Spiral methodology

The spiral model repeats a cycle of planning, risk analysis, engineering and evaluation, and is well suited to projects with genuine technical uncertainty which this project has. It was rejected because much of its structure exists to coordinate risk across a team, and the overhead is not justified for one developer.

7.4. Appendix D: Requirement analysis

The full requirement tables are in section 3.4. This appendix records the use case behind them.

There is one actor, the user. The system has no administrator, and there is no second class of user.

The user can:

- create an account, verify it, log in, log out and delete it
- reset a forgotten password
- store a memory by speaking or typing
- send a photograph or document, and retrieve it later
- ask a question and receive an answer
- ask for a memory to be forgotten, or for everything to be forgotten
- have a reminder set from the sentence they said
- report a reminder as done or as declined
- change when a reminder is due, or how often it repeats
- accept or decline reminders derived from a stored fact
- browse everything stored, and reveal a masked memory
- browse reminders by date
- record money against an account, and ask about a balance
- see today's remaining items and log an expense from a widget

Every one of these is available through conversation except browsing, the widgets and the account actions, which are screens because they are about looking rather than saying.

7.5. Appendix E: Logical data model

The five entities are described in section 3.5.3. This appendix lists the relationships between them and the indexes that exist, since those are part of the design rather than an implementation detail.

From	To	Relationship
User	Memory	One user has many memories. Deleting a user deletes their memories.
Memory	Memory	A memory may have a parent memory, which is how a derived reminder is linked to the fact it came from. Deleting the parent deletes the derived children.
User	Account	One user has many accounts.
Account	Account	An account may name another account that pays it off, such as a card paid from a current account.
User	TxCategory	One user has many categories. A category name is unique per user and per direction of flow.
Account	Transaction	One account has many transactions.
TxCategory	Transaction	A transaction may have one category, or none.
Transaction	Account	A transfer names a second account, which is what makes it a transfer rather than two separate records.

Table 7: Relationships between entities.

Table	Index	Query it serves
Memory	userId, remindAt	A user's upcoming reminders
Memory	notified, remindAt	The sweep looking for reminders that are due and unsent
Memory	userId, isActive, createdAt	Listing a user's memories, newest first
Memory	parentMemoryId	Finding a memory's derived children
Account	userId, archived, sortOrder	The account cards on the budget screen
Transaction	userId, occurredAt	Recent activity
Transaction	accountId, occurredAt	One account's history
Transaction	userId, categoryId	Spending by category

Table 8: Indexes, and the query each one exists for.

7.6. Appendix F: Legal, social and ethical issues in detail

7.6.1. Legal issues

PIPEDA is built on ten fair information principles: accountability, identifying purposes, consent, limiting collection, limiting use and disclosure and retention, accuracy, safeguards, openness, individual access, and the ability to challenge compliance (Office of the Privacy Commissioner of Canada, 2019). Four of them bear directly on this application.

Consent must be meaningful, which means a person has to understand what they are agreeing to. For this application the thing that needs to be understood is not only that data is stored everybody assumes that but that what they say is kept in order to be answered later, and that they can remove any of it at any time.

Limiting collection is unusual here, because the user chooses what to collect. The application does not gather anything in the background: there is no analytics, no location, and no contact list access. What it holds is exactly what somebody typed or said into it.

Safeguards must match sensitivity. Passwords are hashed, transport is encrypted, every request is scoped to the user who made it, and sensitive kinds are masked on screen. The gap is that stored memories are not encrypted at rest with a key the user holds, which would be the appropriate standard for a store deliberately containing credentials.

Individual access is straightforward: a user can see everything they stored, delete any of it, and delete the account as a whole. There is not yet an export.

One further point specific to Canada is worth noting. Section 184 of the Criminal Code makes it an offence to intercept a private communication. This application does not do so speech is converted to text by the phone's own recognition and the audio is never transmitted or retained but any future feature that recorded a conversation rather than a note would need to be looked at against that provision before being built.

7.6.2. Social issues

The dependence question is the one most often raised about tools of this kind. There is a reasonable answer to it: the application is aimed at information that people were never able to hold reliably in the first place, such as a document number or a dose. Offloading those is not the same as offloading the act of remembering. But the answer is a reasonable position rather than a proof, and somebody who disagrees is not being unreasonable.

7.6.3. Ethical issues

The decision to hide something is an ethical decision as much as a design one, because the application is deciding on the user's behalf what is embarrassing or dangerous. It errs towards hiding.

The last ethical point concerns how accuracy is reported. It would be straightforward to quote the single best number this project has produced and stop there. The honest position is that the application is measured against cases written by the person who built it, which flatters it, and that its known weak points are stated rather than left out. For an application that people may trust with a medication dose, that seems like the minimum standard.

7.7. Appendix G: Software Requirement Specification

7.7.1. Introduction

This specification describes the requirements for Remember Me, a mobile application that stores information given in natural language and returns it in natural language.

7.7.2. Intended audience

This document is intended for anyone who needs to understand what the application does and why it is built the way it is: a developer joining the project, a reviewer assessing it, or the author returning to it later. A reader who only wants to know what the application does should read Chapters 1 and 4. A reader who needs to change it should read Chapter 3 and this appendix.

7.7.3. Project scope

In scope: storing memories in natural language by voice, text or photograph; retrieving them by question; reminders including deadlines, recurrence and derived reminders; masking of sensitive memories; personal money tracking; home screen widgets; and account management including deletion.

Out of scope: sharing memories between users; collaboration of any kind; an administrator role; a web application; and synchronisation of local-only memories between devices.

7.7.4. System features

The system features are the functional requirements listed in section 3.4, grouped as: account management (FR-01 to FR-03, FR-24), capture (FR-04 to FR-06, FR-18), retrieval (FR-07 to FR-09), reminders (FR-10 to FR-16), presentation (FR-17, FR-19, FR-20), money (FR-21, FR-22) and widgets (FR-23).

7.7.5. User classes and characteristics

There is one user class. Users are assumed to own a smartphone and to be able to speak or type a sentence. No technical knowledge is assumed, and no knowledge of how the application organises anything is assumed, because it does not ask the user to organise anything.

7.7.6. Operating environment

The mobile application runs on iOS and Android. The backend runs as a single service against a managed database.

7.7.7. Design and implementation constraints

i) Voice must be converted to text on the device. Audio must not be transmitted.

ii) Money must be stored in minor units as integers.

7.7.8. Assumptions and dependencies

The application assumes that the device has an internet connection when storing or recalling in natural language, that the phone's clock and timezone are correct, since every date resolution depends on them, and that the user permits notifications, without which reminders cannot be delivered.

It depends on an email service for verification and password reset messages, and on the phone platform for builds and notifications.

7.7.10. Non-functional requirements

The non-functional requirements are listed in section 3.4.

7.7.11. Safety and security requirements

i) A memory must never be returned to a user who does not own it. This is checked on the server for every route that takes an identifier.

ii) A password must never be stored or logged in a form that can be read back.

iii) A token issued before a password reset must stop working, so that a reset actually evicts somebody who has stolen a session.

iv) Repeated wrong password attempts must be limited per account, so that the limit cannot be avoided by changing address.

v) The reminder sweep must not be able to send the same reminder twice, even with several server instances running.

vi) Every external call must have a timeout.

vii) No API key may be shipped inside the mobile application.

7.9. Appendix I: Screenshots of the system

The screens described in Chapter 4 are shown here together, so that the application can be looked at as a whole rather than a flow at a time.

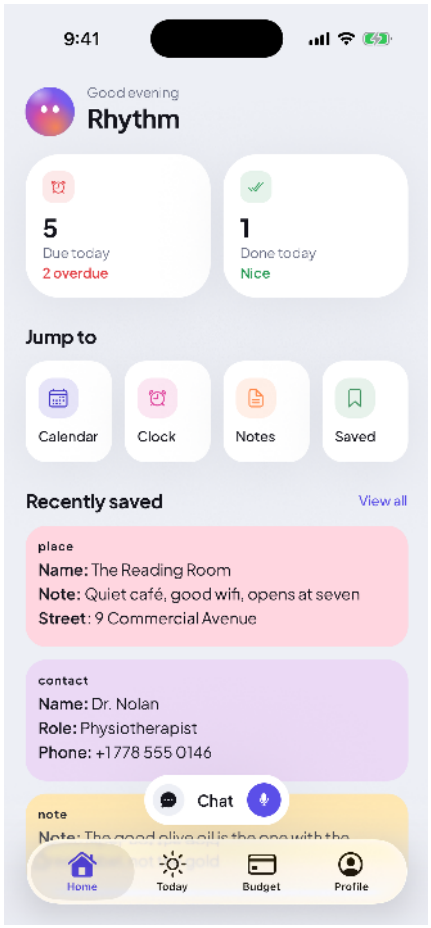

Figure 19: Home.

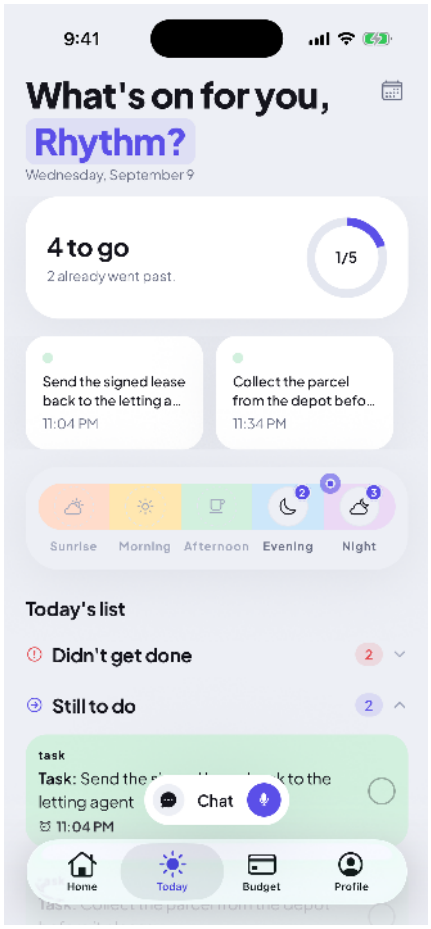

Figure 20: Today.

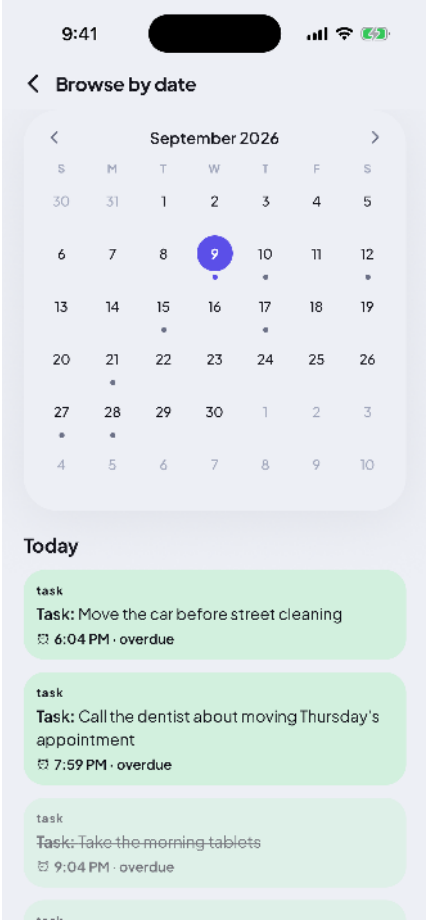

Figure 21: Browsing by date.

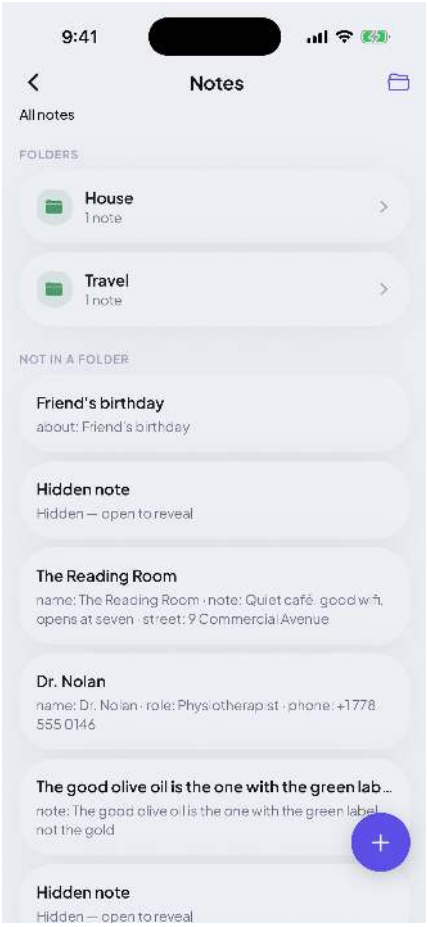

Figure 22: Notes and folders.

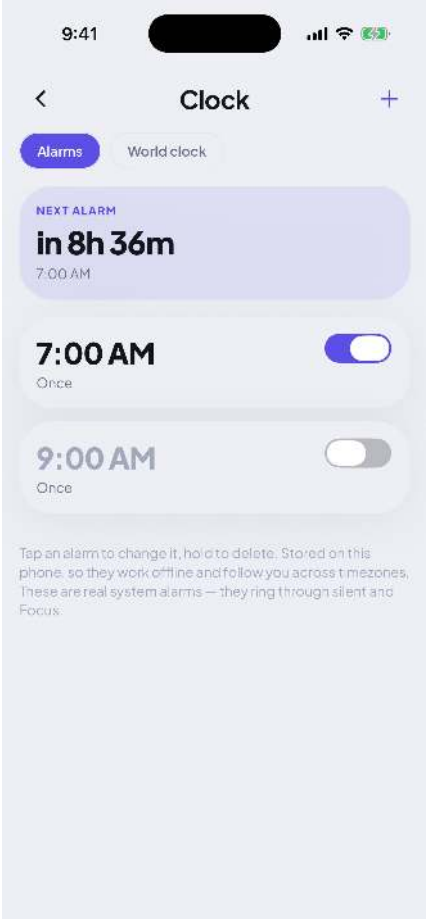

Figure 23: Alarms.

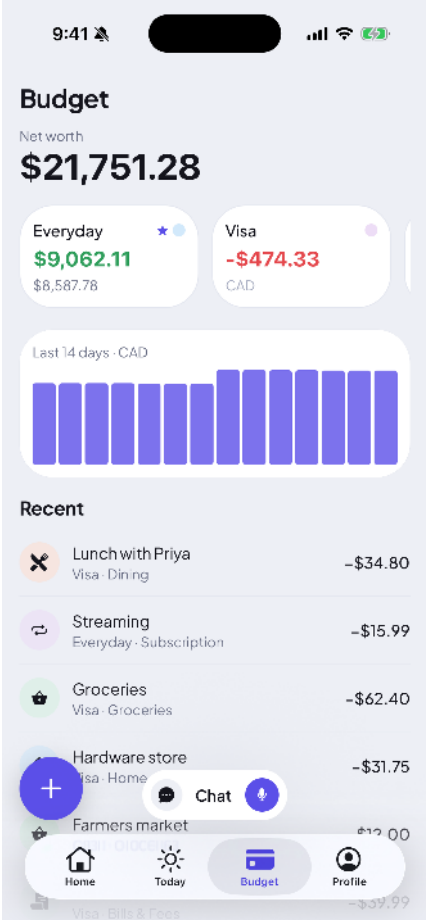

Figure 24: Budget.

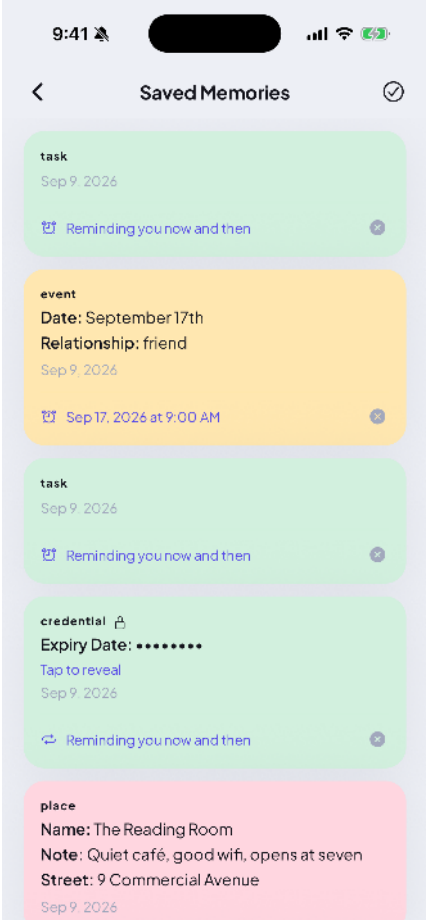

Figure 25: Everything saved.

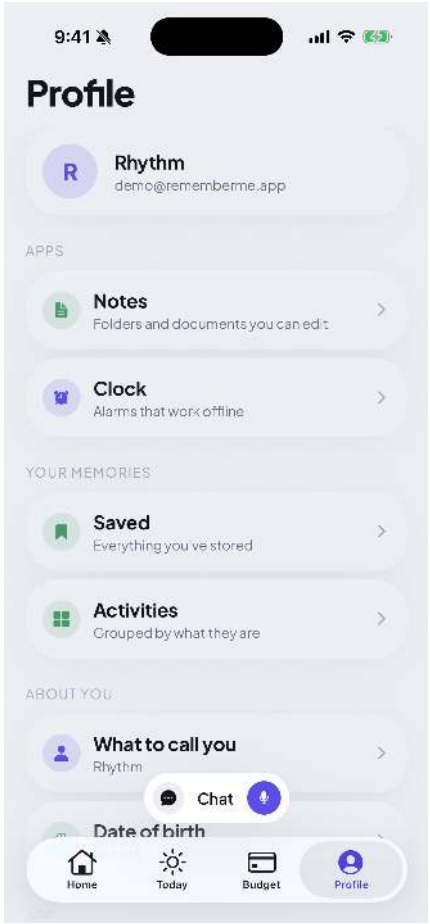

Figure 26: Profile.